UNIVERSITÄT
BERN

TA Session 3

Anthony Gillioz
Institute of Computer Science – University of Bern, Switzerland

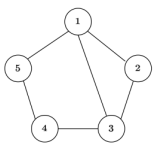
Contact: anthony.gillioz@unibe.ch

UNIVERSITÄT BERN

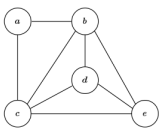
Theoretical Tasks

Task 1

1. Compute eigendecompositions of the two nearly isomorphic graphs g_1 and g_2, compute $\bar{U}_{g_1}\bar{U}_{g_2}^T$ and find assignment with LSAP solver (use numpy and scipy). Write down all intermediate matrices that result from the whole process.



g_1



g_2

- You can use Python (Numpy, Scipy) to solve this exercise.
- Expected: Adjacency matrices, corresponding Eigendecompositions, $\bar{U}_{g_1}\bar{U}_{g_2}^T$, and the assignment found by LSAP solver.

UNIVERSITÄT BERN

Theoretical Tasks

Task 2

2. Enumerate the basic steps of (a) spectral graph matching and (b) continuous graph matching in a table with two columns. Elaborate on the fundamental differences and similarities of the two approaches for graph matching

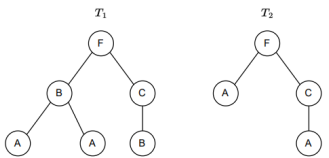
- First table contains the basic steps of (a) and (b).
- Second table contains the differences and similarities between those two approaches.

UNIVERSITÄT BERN

Theoretical Tasks

Task 3

3. Compute the tree edit distance between the two trees T_1 and T_2. The cost of insertion or deletion of a node n is defined as $c(n \to \epsilon) = c(\epsilon \to n) = 1$. The cost of substituting one node to another is 0 if both labels are identical and 1 otherwise.



- Write down all the intermediary matrices.
- Refer to Fig. 6.9 in the lecture notes for an example of what the solution should look like.

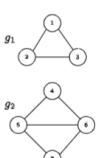
UNIVERSITÄT BERN

Theoretical Tasks

Task 4

4. Given are two graphs g_1 and g_2 with three and four nodes, respectively. Assume that all nodes of both graphs are embedded in a vector space (e.g., by means of spectral embedding as discussed in Section 5.1.2) and we already have computed the pairwise dissimilarities between all nodes (see Table D below).

   Perform an agglomerative hierarchical clustering based on the dissimilarity matrix D using single linkage (refer to the Appendix for details). To this end, sketch the corresponding dendrogram and then analyze the clustering result and identify the clustering so that at most three nodes are present per cluster. Formalize the resulting many-to-many node mapping defined by this particular clustering.



D

	1	2	3	4	5	6	7
1	0	0,9	1,43	0,81	0,87	1,32	0,1
2		0	1,15	0,23	0,15	1,1	0,81
3			0	1,45	1,21	0,2	1,67
4				0	0,25	1,2	0,85
5					0	1,13	0,8
6						0	1,23
7							0

- Perform the hierarchical clustering and draw the dendrogram you obtain (see Appendix of the exercise sheet).
- Based on this clustering draw the many-to-many node mapping.

Implementation Tasks

In this implementation task, you have to implement two algorithms. The first algorithm is String Edit Distance (SED) (Chap. 6.1). The second implementation task is to implement a genetic algorithm (Chap. 5.3).

Remarks:

- The entire code must be contained within the file `PR_lecture/Exercise_3/ex3_a.py` and in `PR_lecture/Exercise_3/ex3_b.py`.
- You are allowed to modify the code as much as you want, including changing function signatures, creating new functions or classes, and so on.

Implementation Tasks - a
Remarks

In the first implementation task, you have to implement the SED algorithm (Alg. 6 in the lecture notes).

- The alphabet Σ is restricted to lowercase and uppercase Roman letters, represented by the set $\Sigma = \{a, b, c, \ldots, z, A, B, C, \ldots, Z\}$.
- For any letter l in Σ, the cost of insertion or deletion of l is defined as $c(l \to \epsilon) = c(\epsilon \to l) = 1$.
- The cost of substituting one letter l for another letter l' is 0 if both letters are identical and the same case (e.g., 'A' == 'A', 'v' == 'v'), 1 if the letters are the same but have different capitalization (e.g., 'a' == 'A', 'V' == 'v'), and 2 if the letters are completely different.

UNIVERSITÄT BERN

Implementation Tasks - a

Idea of code structure

```python
def main():
    # Load list of words/texts to compare from 'data/texts.txt'

    # Clean and split the words

    # Compute the string edit distance between all pairs of loaded words

    # Save the GEDs in './results/SED_results.csv'
```

UNIVERSITÄT BERN

Implementation Tasks - a

Idea of code structure

```python
# Define Cost functions


def sed(string1: str, string2: str) -> float:
    """
    Compute the string edit distance between the two given strings.

    Args:
        string1: A string sequence
        string2: A string sequence

    Returns:
        String edit distance between the two input strings.
    """
    return 0
```

Implementation Tasks - b

Remarks

In the second task, your goal is to code a genetic algorithm to find a permutation matrix P that minimizes the following cost between the adjacency matrix A_1 of graph g_1 and the adjacency matrix A_2 of g_2 $||A_1^T P - P A_2^T||_2^2$.
Experiment with various parameters of your algorithm, such as the number of chromosomes (N), mutation rate, crossover rate, and number of iterations, to find the permutation matrix P' that satisfies the condition $||A_1^T P' - P' A_2^T||_2^2 \leq 4.0$.

UNIVERSITÄT BERN

Implementation Tasks - b
Remarks

- Use Alg. 5 from the lecture notes as a basis for your implementation.
- For any steps that are not specific to the graph-based method (e.g., how to perform the chromosome selection), you are free to implement any mechanism of your choice.
- You are free to choose which crossover mechanism you want from the lecture notes: partially mapped crossover (PMX), cycle crossover (CX), and order crossover operator (OX).
- you are free to choose which mutation mechanism you want from the lecture notes: exchange mutation operator (EM), insertion mutation operator (ISM), displacement mutation operator (DM), and simple-inversion mutation operator (SIM).

UNIVERSITÄT BERN

Implementation Tasks - b

Idea of code structure

```python
def ex3_b():
    # Set random seed
    random.seed(42)
    np.random.seed(42)

    # 1. load the data
    graphs = load_all_graphs('./data')

    # 2. Run the Genetic algorithm on the adjacency matrices of the data
    # 2.1 Tweak the parameters of the genetic algorithm to obtain a fitness score <= 4.0

    # The fitness score of your solution has to be <= 4.0
    assert score <= 4.0
```

UNIVERSITÄT BERN

Implementation Tasks - b

Idea of code structure

```python
class GeneticAlgorithm:

    def __init__(self, ...):
        pass

    def create_initial_population(self, ...):...

    def evaluate_fitness(self, ...):...

    def selection(self, ...):...

    def crossover(self, ...):...

    def mutation(self, ...):...

    def run(self, A_1: np.ndarray, A_2: np.ndarray) -> (float, np.ndarray):
        """
        Run the genetic algorithm to find the isomoprhism (permutation matrix P)
        between the adjacency matrices of graph 1 and graph 2.

        Args:
            A_1: Adjacency matrix of graph 1
            A_2: Adjacency matrix of graph 2

        Returns:
            A tuple with the fitness score of the best permutation matrix and the corresponding permutation matrix.
        """
        return 0, None
```

UNIVERSITÄT BERN

Implementation Tasks - b

Idea of code structure

```python
def fitness_func(A_1: np.ndarray, A_2: np.ndarray, P: np.ndarray) -> float:
    """
    Compute the fitness score F(P) = ||A_1P - PA_2||_2^2

    Args:
        A_1: Adjacency matrix of graph 1
        A_2: Adjacency matrix of graph 2
        P: Permutation matrix

    Returns:
        Fitness score of the given permutation matrix

    """
    return np.linalg.norm(A_1 @ P - P @ A_2, ord=2) ** 2
```